PROFILING USERS FROM GEOLOCATION DATA

A 5IF *Distributed Social Networks* project

by Victor Capelle, Josquin Cornec, Xavier Dussert-Vidalet, Timothée Gonnet, Lucas Ouaniche-Herbin & Louis Sugy

Introduction

Team

Xavier Dussert-Vidalet	**Scrum Master**
Lucas Ouaniche-Herbin	**Product Owner**
Louis Sugy	Software Architect & Technical Director
Victor Capelle	Data Scientists
Timothée Gonnet	
Josquin Cornec	Technology Intelligence

Context and goals of the project

According to Pew Research Center surveys, 77% of U.S. adults say they own a smartphone in 2017. In Western European countries, the percentage ranges from 58% in France to 79% in the Netherlands according to the same surveys. Connected devices also include cars, wearable devices, etc.

A lot of services have access to the user's location. Even if this legally requires permission from the user, the average user doesn't even know that they allowed the services to access this information, or simply don't realize how harmful the processing of this data can be to their privacy.

These services, in accordance with GDPR, are not supposed to process sensitive data, nor can they sell to their partners information that identifies the user. Here is an extract from Google's privacy policy:

> - We don't show you personalized ads based on sensitive categories, such as race, religion, sexual orientation, or health.
> - We don't share information that personally identifies you with advertisers, such as your name or email, unless you ask us to. For example, if you see an ad for a nearby flower shop and select the "tap to call" button, we'll connect your call and may share your phone number with the flower shop.

However, unless you specify in your settings that you don't want Google to keep your location history, they collect and process all your location data all the time, even when you are not using a geolocation-based service such as Google Maps, as long as the Google Services are installed on your smartphone. What we want to demonstrate with this project is that **location data itself contains a lot of sensitive data**.

Available data

We were given data collected by our teachers from more than a hundred users over a time range of approximately one year, with their agreement to use the data for the purpose of science and education. The use of this data was supervised by the researchers who collected it, and we removed all our copies after demonstrating the project results. The present report doesn't show any precise data that could help identifying our users in any way.

The data contains GPS coordinates collected frequently (one or two per second when the phone was on) associated with the ID of an anonymous user.

Our product

We wanted to develop a tool as a practical result of our research, and in order to prove that it is possible within only a few weeks and with no previous knowledge on the matter, to achieve a decent quality of user profiling based on simple location data.

Our product is intended to provide the full processing pipeline so that the tool's user (called *the attacker* in this document to distinguish them from the tracked smartphone users) can directly query meaningful information from raw location data.

The actual tool is a command-line interface (CLI), though to allow further addition of a graphical interface the code has been structured such that it can be compiled into either the CLI or an application programming interface (API).

The following features were implemented:

- Finding the home of a user (and their possible second home, as some people live in a different place during the week and weekends, or don't live with their partner).
- Finding the workplace of a user.
- Finding the type of places where a user often goes (e.g restaurant, school if they have kids, library, etc).

Features that were planned but not implemented because of the time constraints:

- Guessing the user's age, gender, religion.
- Identifying relations between the users (our set of users was too small though).

Organization

We use an Agile organization. We don’t have a real client but we define ourselves our goals and changes of direction of the project.

The main working steps are research and then implementation.

Research

- Bottom-up research from the given data: explore the data with KNIME to discover relevant information that could be deduced from the data.

- Top-down research on the given data: think about what we can try to find in the data, including non-trivial information that could be harder to find.

- Research of other data sources: find other relevant data sources that could bring more semantics to the information found in the previous steps, e.g identify a place from its GPS coordinates.

- Literature research (state of the art): reading multiple articles such as theses which give examples of algorithms used to estimate points of interest or some explicit use cases where attacks happened.

Implementation

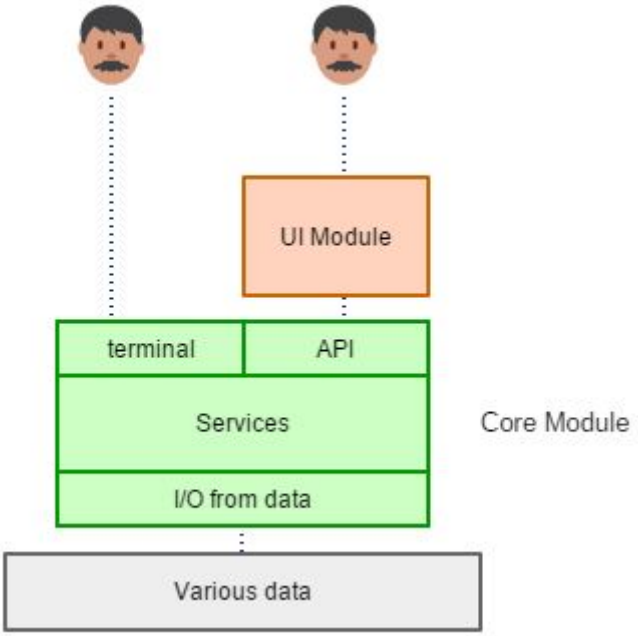
- Specify the architecture.
- Create the structure of the project, setup our tools.
- Implement the backend: data, algorithms and services.
- Implement the frontend: workflows, command-line interface.

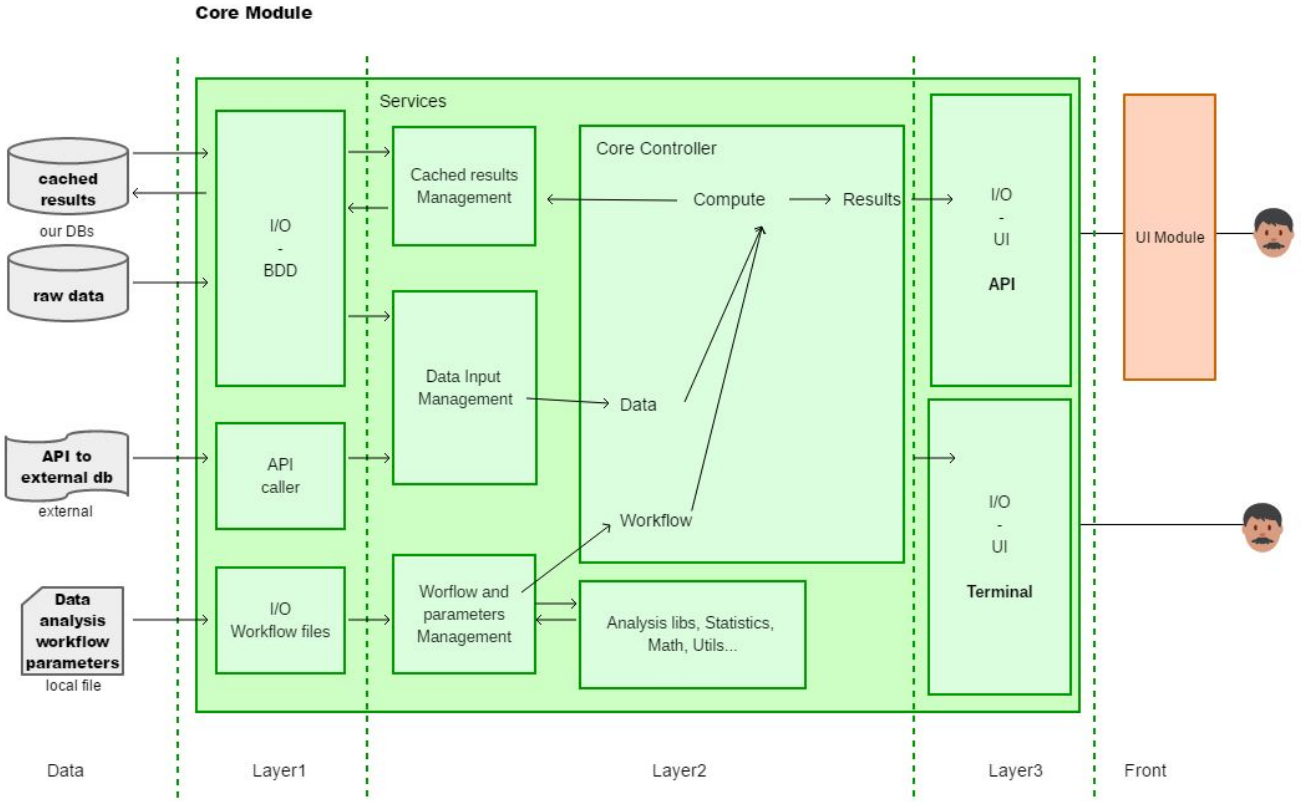
We always keep in mind possible evolutivity : Agile objectives and future targets.

Technical overview

Architecture specification

The application is organized in layers, to make group work easier, improve modularity (including allowing multiple endpoints), and avoid binding the algorithms to the format of the data.





Technology

For the final product, we made the following technology choices:

- C++ as the main programming language (powerful enough for the processing of large data, and everybody in the team has already use that language).
- InfluxDB as the database to store the raw data (it is a time series database very well suited for our use case, i.e the fast retrieval of millions of measurements associated to timestamps and tags - here, the user id).
- Python scripts to pre-process the data and insert it into the database.
- Bash script to send requests to the OpenStreetMap API
- The OpenStreetMap-based Overpass API (very powerful query language and it guarantees to keep ourdata confidential).
- C++ libraries: *JSON for Modern C++*, *influxdb-cpp*, *cppformat*, *R-Trees*, *cpp-httplib*

We also used the following tools:

- GitLab version-control system and task management (board, issues).
- Konstanz Information Miner (KNIME) for early data exploration.
- GNU Make and the GNU Compiler Collection (GCC) for compilation.

Algorithms

Identifying points of interest (clustering)

We have a set of points extracted from the Influx database, and we want to start recognizing patterns in our users' behaviour. In this part, we are focusing on finding points of interests, which are locations where the user goes repeatedly or where he stays a lot. So we are using a clustering algorithm, based on DBSCAN. This algorithm, dj-cluster, uses a join-based clustering to tackle the performances issues that DBSCAN can have. (*Discovering Personal Gazetteers: An Interactive Clustering Approach, Zhou et al*).

In order to improve the performance of this algorithm, we based our sample data that we got from Influx on a R-Tree to improve research time, and fusion-time between clusters.With this, we can achieve the n log(n) complexity that this algorithm can have.

Processing the clusters

We have multiple functions to process the clusters and make their analysis easier:

- `cluster_centroid`: gets the centroid of a locations' cluster (projects the points in 3D, finds the centroid and projects back into latitude and longitude).

- `divide_cluster`: takes a points cluster and divides it into unique visits (according to parameters given in the configuration file).
- `get_cluster_hours_spent`: finds the time spent in a given point cluster corresponding to a unique visit.

Finding the home and workplace

Home and workplace are - supposed to be - among the places the user is the most. But how to distinguish them ? Some people spend more time at home, others at work. So using the time spent in each cluster is not enough.
Assuming the *home is where the person sleeps,* we defined the house as the cluster where the person has spent the most time during the *critical night*: a part of the night when we are sure a lot of people sleep. We arbitrarily chose 01h00AM - 06h00AM. We can observe that there is often more than 1 cluster where a person is a lot during *critical nights*.
Our find_house function returns 2 clusters corresponding to “houses”.

For the workplace we used the same principle with *critical working hours*. Workplaces results seem to be less relevant, the algorithm could be improved using the day of the week or inferring working patterns in the movements.

Finding frequent place types

In order to find frequent place types from points of interests, we use the Overpass API. We perform the API calls asynchronously while doing other calculations, in order to avoid having the CPU idle while waiting for the response.

In OpenStreetMap, special places are often identified by special tags, which are in fact key-value couples, such as `amenity=university`. In the configuration file, we list the keys that are interesting to us. Then we construct an overpass query to find the elements with tags among these keys and keep the corresponding values.

The Overpass query looks like:

```
[out:json][timeout:10];
(way(around:10,45.xxxx,4.xxxx)[~'^(amenity|leisure|tourism|building|shop|health
care|aeroway|craft|historic|landuse|military|natural|office|public_transport|sp
ort)?$'~'.'];
node(around:50,43.4093,3.68767)[~'^(amenity|leisure|tourism|building|shop|healt
hcare|aeroway|craft|historic|landuse|military|natural|office|public_transport|s
port)?$'~'.'];);
out tags;
```

We then count the number of times every tag appears in all the points of interest and are able to display which tags appear most. See an example of execution in the next part.

Results

Use guide of our final product

The settings are not read at compile time but at run time, thanks to the file config.txt which takes the following form:

```
# DJ parameters
eps=0.0002
min_pts=20

# Radius of search of an amenity around coordinates
search_radius=30.

#Night hours (integers)
night_start=1
night_end=6
work_start=11
work_end=16

# Interval in minutes between two points to consider them in different
# unique visites
sep_minutes=10.

# OpenStreetMap attributes to consider when tagging a position
place_types=amenity,leisure,tourism,building,shop,healthcare,aeroway,craft,hist
oric,landuse,military,natural,office,public_transport,sport

# Minimum time interval between two points
precision_sec=3600.
```

Then, when launching the CLI:

First you have to specify the user (and optionally the time range) to work on :

```
>>> load id [t1 t2]
```

Find where the user lives. This gives 2 main living-points (and another ignoring durations):

```
>>> house
```

Find where the user works. This give a place (and another ignoring durations):

```
>>> workplace
```

Find tags about the places:

```
>>> frequent-places
```

Show the centroids of points of interest found:

```
>>> show-clusters
```

To leave the application:

```
>>> exit
```

Example of an execution (sensible data removed)

Here is an example of execution of our tool. We removed the exact locations, and estimated that the tags shown here are not enough, alone, to identify any specific person.

```
>>> load 4
Got 1209727 points
Kept 2190 points
Computing points of interest…

>>> house
This person seems to sleep the more at : 45.xxxx 4.xxxx
And at : 45.xxx 4.xxxx
(WITHOUT DURATION : 45.xxx 4.xxxx and 45.xxxx 4.xxxx)


>>> workplace
This person seems to work at : 45.xxxx 4.xxxx
(WITHOUT DURATION : 45.xxxx 4.xxxx)

>>> frequent-places
1. apartments
2. residential
3. school
4. kindergarten
5. garages
6. parking
7. restaurant
8. sports_centre
9. gymnastics
10. playground
```

Conclusion

This work helped us to understand how geolocation data can be used to guess sensitive information about people. It taught us that it is virtually impossible to filter out personal and sensitive information from the collected data. Most smartphone users are not aware of how harmful it can be to allow companies to collect that kind of data.

It also showed that as Software Engineers we will have to deal with these concerns when collecting data from the users, and it is important that we understand the responsibility that we have when dealing with data that potentially contains sensitive information, and especially what do we do to make sure that this data is not accessed by someone with malicious intentions (how secure is the system, do we collect and store more data than we actually need ?).

On a technical point of view, this project was challenging as we started with a huge quantity of raw data, and came up with solutions to profile the users from this data by using the right technology to work efficiently with big data, maths to identify the points of interest, and queries to the Overpass API to exploit actual geographical data and identify what the points of interest correspond to.